基于多核 FPGA 的压缩文件密码破译 *

陈晓杰 [1]，周清雷 [1]，李　斌 [2]

(1. 郑州大学 信息工程学院，郑州 450001; 2. 信息工程大学 数字工程与先进计算国家重点实验室，郑州 450001)

摘　要：目前，破解 WinRAR 传统方法是使用 CPU 和 GPU，而潜在的密码空间非常大，需要更高性能计算平台才能在有限的时间内找到正确的密码。因此，采用四核 FPGA 的硬件平台，实现高效能的 WinRAR 破解算法。通过在全流水架构下增加预计算和保留进位加法器结合的方法优化 SHA-1 算法，提升算法吞吐率；利用状态机的控制优化数据拼接，提升算法并行性；同时，采用异步时钟和多个 FIFO 缓存读写数据优化算法整体架构，降低算法内部的耦合度。实验结果表明，最终优化后的算法资源利用率为 75%，频率达到 200 MHz，4 bit 长度的密码破译速度为每秒 102 796 个，是 CPU 破解速度的 100 倍，是 GPU 的 3.5 倍。

关键词：信息安全；密码破译；FPGA；高性能计算；WinRAR

中图分类号：TP309.7　　doi: 10.19734/j.issn.1001-3695.2018.06.0478

Password recovery for compressed file based on multi-FPGA

Chen Xiaojie[1], Zhou Qinglei[1], Li Bin[2]

(1. *School of Information Engineering*, *Zhengzhou University*, *Zhengzhou* 450001, *China*; 2. *State Key Laboratory of Mathematical Engineering & Advanced Computing*, *Information Engineering University*, *Zhengzhou* 450001, *China*)

Abstract: At present, the traditional method of cracking WinRAR is to use the CPU and GPU, but the potential password space is very large which requiries a higher performance computing platform to find the correct password within a limited time. Therefore, this paper uses the hardware platform of multi-FPGA to achieve a high-performance WinRAR crack algorithm. The SHA-1 algorithm was optimized by adding pre-calculation and carry saving adder under the full-pipeline architecture, so as to improve the throughput of the algorithm. And the use of state machine control to optimize data splicing, to improve algorithm parallelism. At the same time, it used the asynchronous clock and multiple FIFO buffers to read and write the overall structure of the data optimization algorithm to reduce the coupling within the algorithm. The experimental results show that the final optimized resource utilization rate is 75% and the frequency reaches 200MHz. The 4-byte password deciphering speed is 102, 796 per second, which is 100 times faster than CPU and 3.5 times faster than GPU.

Key words: information security; password cracking; FPGA ; high-performance computing; WinRAR

chinaXiv:201811.00131v1

0　引言

压缩软件 WinRAR 是一款 RAR 压缩文件管理器，具有压缩、纠错、加密等多种功能，并且支持多种压缩格式的解压。其压缩速度快、压缩效率高等多种优势得到了个人、机构、组织等的广泛使用。WinRAR 密码加密使用对称加密算法 AES 和单向不可逆算法 SHA-1，这两种算法在目前的计算能力下能保证很高的安全性。但是，这也使需要对 WinRAR 进行快速密码破译的信息安全和计算机取证带来了一定的困难。

目前，破译 WinRAR 的密码采用字典+穷举的方法，但随着密码长度的增加，可能的空间呈指数倍增长。近期研究发现用户密码服从 Zipf 分布，而不是长期假设的均匀分布[1]。这一发现对“用户密码”的研究有很重要的指导意义。在密码破译中也能够暂时性的缩减密码空间。文献[2,3,4]中详细研究讨论了“口令猜测算法”及“口令猜测模型”，并利用社会工程学等方法获得针对性的信息，从而分析出某些高概率口令的特征，或者针对某一用户的高概率口令。因此，利用这些高概率口令可以破译很多用户密码，缩减了密码空间，提高破译效率。虽然密码空间能够缩小很大一部分，但是一个高效能的计算平台对进一步提高破译效率具有很重要的作用。

现有的破解方法主要有依托于 CPU 平台的密码软件破解，破解速度慢。基于 OpenCL 和 CUDA 架构的 GPU 平台破解，

收稿日期：2018-06-28；**修回日期：**2018-08-24　**基金项目：**国家重点研发计划资助项目（2016YFB0800100）；国家自然科学基金面上项目（61572444）

作者简介：陈晓杰（1993-），男，河南武陟人，硕士研究生，主要研究方向为信息安全（740248499@qq.com）；周清雷（1962-），男，河南郑州人，教授，博导，博士，主要研究方向为信息安全、自动机理论及计算复杂性理论；李斌（1986-），男，河南郑州人，博士研究生，主要研究方向为信息安全、高性能计算.

ChinaXiv合作期刊
chinaXiv:201811.00131v1

由于其受到访存的限制，影响了计算性能的提升，而且功耗较高。因此，需要更高效的计算平台来提高破解的效率。现场可编程门阵列（field programmable gate array, FPGA）经过近 30 年的发展，已成为实现数字系统的主流平台之一，在以太网交换机传输数据、图像压缩、模板计算的加速等方面都得到了较好的应用[5-7]。将 FPGA 用于 WinRAR 的密码解密中，利用流水线并行工作的方式，可实现 WinRAR 密码破解算法的高效性。本文的主要工作是基于四核 FPGA 硬件平台，提出对 SHA-1 算法关键路径的优化，并通过状态机的控制优化数据拼接。算法的整体架构采用异步时钟使核心模块运算频率最大化，并使用多个独立时钟双端口 Block RAM 实现的 FIFO 配合全流水线进行数据缓存读写。最后通过实验，对比不同平台 WinRAR 破解算法的性能，表明本文优化后的算法性能有明显的提升。

1　算法分析

WinRAR 破解算法主要分为四步，具体流程如下：

a)提取特征串。从压缩文件中提取盐值 salt 和哈希值 digest，用于输入计算。

b)数据拼接和 SHA-1 运算。将可能的密码（宽子节）、8 Byte 的盐值、3 Byte 的 num 依次进行拼接。其中，num 表示拼接的次数。整个解密过程一共需要拼接 262 145 次，每拼接 64 字节需要进行一次 SHA-1 运算。每拼接 16 384 次需要进行一次额外的 SHA-1 运算，用于获取密钥 key 和 iv 值。

c)AES 运算。将 digest 与 key 做 AES 运算，得到 aes_r，并将 aes_r 与 iv 异或,得到 xor_r。

d)比较。将 xor_r 和一个固定值进行比较，判断密码是否正确。算法结构如图 1 所示。

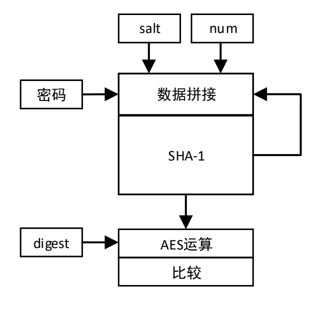


图 1　解密算法结构图

Fig.1　Decryption algorithm structure

在整个算法中计算量最大的就是数据拼接操作和 SHA-1 运算，并且随着密码长度的增大，需要进行 SHA-1 运算的次数呈线性增长，计算需要的时间也呈线性增长，如图 2 所示。图中以计算一个 4 位长度密码的时间为基准单位来衡量其他长度密码的计算时间。

从算法分析中可以知道对数据拼接操作和 SHA-1 的优化对算法的整体性能提升有至关重要的作用。文献[8]中在 GPU 平台下在 CPU 端预先生成一个拼接表，再将此表一部分放到共享内存中，为 GPU 端进行计算，最终在 w8000 GPU 上破解 4 位长度密码的性能达到了每秒 10100 个，比商业破解软件 AccentRAR 在 GPU 上快了 2 倍多。文献[9]中在 GPU 平台下通过增加三个数组的方式减少状态跳变和数据链接所需要的时间，再结合 GPU 本身的优势进行优化，比 AccentRAR 在 CPU 上的速度高 43 到 57 倍。文献[10]是基于 CUDA 架构对整体算法进行优化，提高了性能。文献[11]通过对口令空间采用一种基于并行随机搜索的新方法生成字典，提高破解的成功率。在 FPGA 平台上，对于 SHA-1 算法的优化，文献[12,13,14]通过使用保留进位加法器（Carry Save Adders，CSA）的方式减少关键路径的延时。文献[15]通过增加 P、Q、R 三个中间变量进行预计算减少延时。文献[16]基于循环展开方法的 40 级流水线提高算法的吞吐率。文献[17,18]利用循环展开和预计算的方式减少关键路径的长度和循环的轮数，达到高速运行。

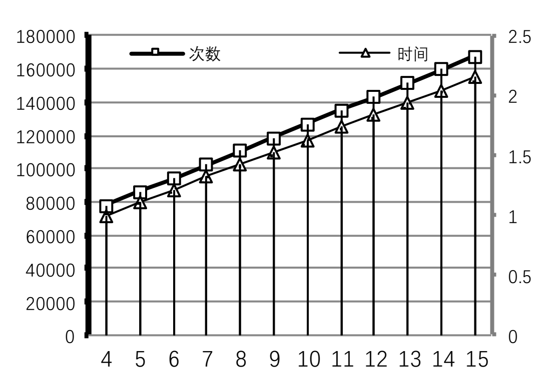


图 2　不同密码长度对应的 SHA-1 次数及相对时间

Fig.2　The number of SHA-1 and relative time

上述研究 WinRAR 破解性能的提升主要集中在 GPU 平台，由于 GPU 要和 CPU 共享内存，限制了性能的提升，且功耗较高。对于 FPGA 平台，SHA-1 算法的优化仍然有很大的提升空间。因此，本文的研究基于低功耗的四核 FPGA 硬件平台，通过优化数据拼接操作和进一步优化 SHA-1 算法，提升算法效率。

2　算法优化及实现

2.1　SHA-1 算法优化

SHA-1 算法自 1995 年修订发布以来，广泛应用于数字签名、传输层安全、安全电子交易、无线局域网等信息安全领域。SHA-1 加密流程为补位、初始向量值、80 轮循环运算、自加运算。其中 80 轮运算的作用是更新 A、B、C、D、E 五个值，并作为最后的输出，每个值 32 位，一共 160 位，而 A 值为 $S_5(A)+f_t(B,C,D)+E+W_t+K_t$，t 代表轮数，$f$ 是逻辑函数，K 是常数，W 的值为 $S_1[W_{(t-3)} \wedge W_{(t-8)} \wedge W_{(t-14)} \wedge W_{(t-16)}]$，S 是循环函数。A 值是运算的最长路径，也是整个运算的关键路径。因此，优化 SHA-1 算法的关键路径决定了整体的运行效率。

2.1.1 SHA-1 算法优化方法

算法的吞吐量与频率、流水线级数成正比[21]，因此本文采用预计算与保留进位加法器策略减少关键路径的延时，提高算法的频率，并采用全流水线实现算法，从而能够提高算法的吞吐量。SHA-1 算法核心迭代需要 80 轮运算，且每轮运算需要上轮运算的结果，运算过程中每轮运算互不干扰。同时，在进

ChinaXiv合作期刊

行密码尝试中，每个密码的计算又是相互独立的。因此，可以采用并行全流水线方法实现，80 轮运算需要 80 个时钟周期，再加上预计算与输出一共 82 级流水线。预计算通过增加寄存器 G，使关键路径变为两个加法运算和位运算，减少关键路径的延时。FPGA 适合于位运算，而加法运算延时比位运算高，保留进位加法器 CSA 策略能够减少加法运算，最小化关键路径长度，保证流水吞吐率[19]。CSA 运算如下，其中 a、b、c 为 n 位二进制数：

$$S(a,b,c)=a\wedge b\wedge c$$

$$Ca(a,b,c)=[(a\&b)|(b\&c)|(a\&c)]<<1$$

$$CSA(a,b,c)=S(a,b,c)+Ca(a,b,c)=a+b+c$$

通过增加位运算来减少加法运算，在关键路径运用保留进位加法器使加法运算变为一个从而来减少延迟。因此，在全流水线架构下增加预计算和保留进位加法器策略能够降低延时，提高算法效率。

2.1.2 SHA-1 算法优化实现

在具体实现算法时，分为三个主要模块协同工作实现全流水线技术，并且在其中引入预计算和保留进位加法器策略。 第一个是 W 模块用于计算每轮 W 的值，需要定义如下寄存器数组：

$$reg\ [31:0]\quad W[64:0][15:0],\ W_next[63:0];$$

$$reg\ [31:0]\quad W_65[14:0];$$

$$reg\ [31:0]\quad W_64[13:0];$$

$$\vdots\qquad\vdots$$

$$reg\ [31:0]\quad W_79;$$

这样定义可以节省 3840 个寄存器资源，其中 W_next[i]由 W[i]计算得到，i 从 0 到 63。数组赋值及传递如图 3 所示。

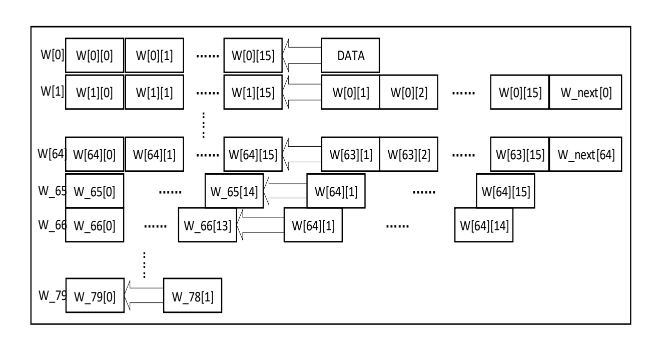


图 3　数组 W 实现全流水线的逻辑结构图

Fig.3　Array W to achieve the full-pipeline logic structure

图 3 中将数组 W 全部展开，使数据传递的过程更加清晰，输入数据 DATA 经过 80 个时钟周期完成 W 的全部计算。单个时钟周期内，数组之间并行处理，且相邻数组数据串行传递，从而实现全流水线的计算，提高了时间效率和资源利用率。

第二个是预计算模块。定义 32 位宽的寄存器数组 G[80],在每轮运算时都预先计算 G 的值$W_t+K_t+E_t$，从而减少关键路径的长度。80 轮运算需要计算 80 个 G 的值，且不同数据的值依次传递，实现了数据内的串行，数据间的并行。

第三个模块是数据更新模块，用来更新 A、B、C、D、E 的值，其中更新 A 的值是关键，其更新如下：

$$assign\quad A_next=CSA\left(\{A[26:0],A[31:27]\},f,G\right);$$

通过使用 CSA 更新 A 值，减少了关键路径的延时。因为每轮运算逻辑函数 f 不同，这一部分又分为四个小模块，但是功能都相同，且同样需要实现流水线架构。因此，最终需要实例化 80 份，以消耗资源换取时间效率最大化。图 4 显示了三个模块协同工作实现 SHA-1 算法的全流水线结构。

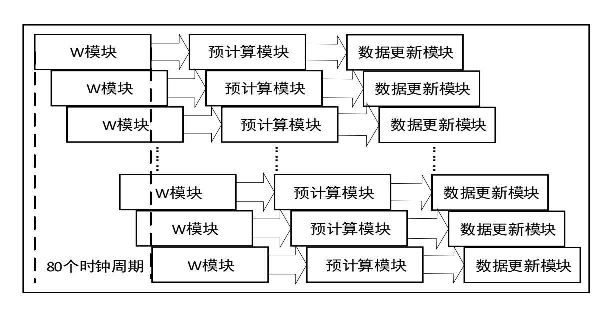


图 4　三个模块实现的全流水架构图

Fig.4　Three modules to achieve the full-pipeline structure

2.2　基于状态机的数据拼接优化

在 WinRAR 密码解密中，数据拼接需要拼接 262145 次，且每拼接 64 字节还要进行 SHA-1 运算，同时还要保存拼接到的位置，以便于下一次拼接从这个位置开始。由于拼接次数多、拼接时间长成为了算法效率提升的瓶颈。SHA-1 算法采用 82 级流水线，数据拼接的数据要输入到 SHA-1 运算，数据拼接需要在 82 个时钟周期内完成 82 组数据的 64 字节拼接操作。因此，本文基于状态机的控制，通过增加位宽为 8 的寄存器数组 pwd_array[39:0]、temp_array[63:0]、flag_array[63:0]、final_array[63:0]使数据拼接能够并行处理，实现流水线的数据传递，提升算法的效率。

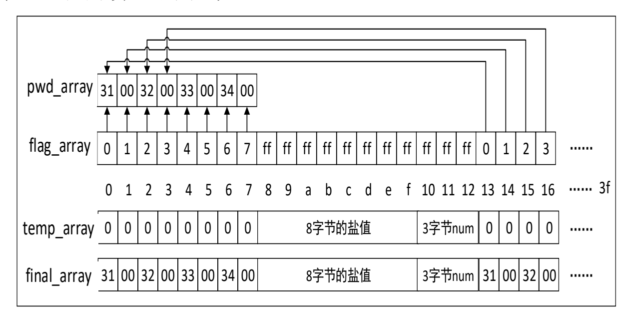


图 5　最终的数据拼接方法

Fig.5　The final data link method

在实现中，数组 pwd_array 用于存储密码，并将密码补位变为 2 字节的宽字节，且支持密码达到 20 位。数组 temp_array 用于存储盐值和 num 到相应位置。数组 flag_array 用于标记口令所在的位置，初始化均设为 0xff，根据状态机来更改数组的值，最后的拼接根据这个值的大小来判断是什么值，并且判断数据所在的位置。相同密码的 temp_array、flag_array 值相同，只需赋值一次，可供流水线 82 个密码共同使用，从而提升时间效率。数组 final_array 则存储最后 SHA-1 运算的输入值。以密码“1234”为例，在图 5 中显示了这四个数组是如何赋值的以及如何拼接的，图中的值都是 ASCII 码 16 进制显示。

chinaXiv:201811.00131v1

ChinaXiv合作期刊
chinaXiv:201811.00131v1

数组的赋值依赖于状态机的控制。状态机控制着数据拼接的开始，初始化，拼接中的数据赋值，状态的保存及状态的返回。具体状态控制如下：

状态 1　对数据进行初始化，包括计算密码的长度 len1，密码和盐值的长度 len2，用于后续判断填充的位置。

状态 2　控制拼接的开始、结束和重返初始状态。

状态 3　判断再拼接一轮是否到 64 字节，如果不超过，则转到状态 4。如果超过，则先将不超过的部分根据 len1 和 len2 的大小给数组 temp_array、flag_array 赋值，并且保存拼接到的位置，同时根据计数器判断给数组 final_array 赋值用于流水线数据传递计算，再转到状态 4。

状态 4：根据 3 状态保存的位置继续下一轮的数组赋值，如果拼够一轮，则返回状态 2，拼接次数 loop 加 1，如此循环，状态转换图如图 6 所示。

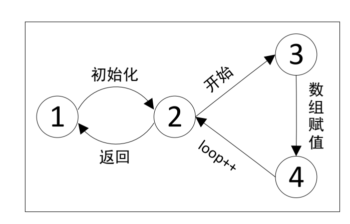


图 6　状态转换图

Fig.6　State transition diagram

通过状态的跳变给数组 temp_array、flag_array 赋值使拼接结构更加清晰，且仅需赋值一次可使流水线计算共同使用，降低了拼接消耗的时间。通过中间数组 flag_array 作为标记，间接链接到密码，使数据的拼接能够并行化，降低结构复杂度，实现流水线，提升算法效率。

2.3　整体算法优化

为了从 FPGA 和强大的计算机辅助设计工具的优势中获益，设计师必须遵循一种高效的设计方法。这种方法基于三个主要原则：控制算法的细化，模块化和最佳的适用性，以及所选择的硬件架构[20]。为了达到这种高效的方法，在整体算法的设计实现时，需要充分利用板卡的 LUT 与 FF 资源，使 FPGA 芯片能够满负荷工作。因此，本文整体算法的设计主要分为上层控制和核心算子两部分。前者通过控制可以实例化多个核心算子，并且对多个算子的结果进行回收，实现多个算子并行工作，从而提升算法的整体性能。上层控制又细分为外部接口模块用于从 CPU 端获取数据，数据管理与分发模块用于将数据提供给多个核心算子。核心算子分为密码穷举模块、核心运算模块和比较模块。模块与模块之间使用 IP Catalog 生成双端口 Block RAM 实现的 FIFO 缓存数据和读写数据。通过 FIFO 在模块之间进行数据的传递，降低算法模块之间的耦合度，提升算法的并行性。同时尽量减少 CPU 与 FPGA 的交互，避免速度的差异致使性能的降低。密码穷举模块实现为掩码字典的方式，在实现时通过上位机传输掩码字典到 FPGA 的口令穷举模块产生相应的口令。掩码字典是一种穷举规则，而在规则中可以包含高概率口令，从而通过人为控制优先测试高概率口令，缩减破译的时间。

细粒度模块结构如图 7 所示。

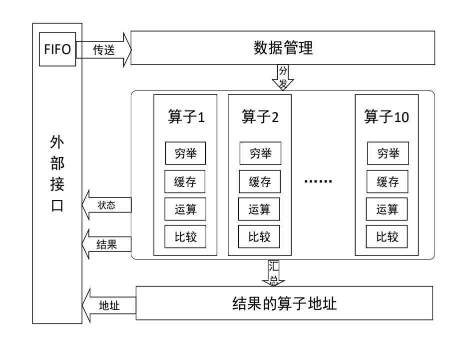


图 7　细粒度模块结构图

Fig.7　Fine-grained module structure

算法的吞吐量、运行速度与时钟频率是正相关，时钟频率越高，运行效率也越高。但是频率过高，容易产生时序阻塞问题，致使 FPGA 布线失败，因此要对时钟频率进行优化。分析整个算法，可以发现计算量最大、计算最复杂的是 SHA-1 运算和数据拼接部分，这两个核心运算部分可以设计成一个高频率时钟，加快核心运算速度。而为这两部分提供数据的上层控制模块和口令穷举模块采用低频率时钟足以满足核心计算需求。通过将时钟域分离，既能充分利用板卡性能，也能最大化降低时序问题，使算法性能进一步提升。

3　实验结果及分析

本文实验的硬件平台是在四核 FPGA 集成加速卡上实现，芯片型号为 XILINX 公司的 XCKU060，其查找表 LUT 资源是 331680，FlipFlops 寄存器资源是 663360。软件平台为集设计、仿真、综合、布线、生成于一体的 Vivado 软件。通过对算法的不同部分进行相应优化，使最终算法在吞吐量、运行速度、资源利用率等方面有较大的提升。

表 1　不同文献 SHA-1 实现性能对比

Table 1　SHA-1 performance comparison

对比文献	流水线级数	最大频率	吞吐量
文献[12]	82	43.08MHz	22.056Gbps
文献[13]	81	272 MHz	139.264Gbps
文献[14]	81	303.3 MHz	155.289Gbps
文献[15]	83	123.3MHz	63.129Gbps
文献[16]	40	163.7 MHz	76.195Gbps
文献[17]	4	150.7 MHz	7.35Gbps
本文	82	470 MHz	240.64Gbps

本文在实现全流水线架构的基础上通过使用预计算和保留进位加法器对 SHA-1 算法进行优化，其优化后的性能如表 1 所示，在表 1 中同时给出了其他文献实现的性能。

通过对比，可以发现本文优化后的性能在频率上达到了 470 MHz，吞吐量达到了 240.64 Gbps，比最高吞吐量的文献[14]还高出了 85Gbps，是文献[16]的 32 倍，性能有了显著提升。为了更进一步的验证优化后的性能，设计了表 2 的 5 种方案在 XCKU060 上实现。

ChinaXiv合作期刊

表 2 不同优化对比方案

Table 2　Different optimization comparison project

方案	不同优化方法
1	不进行任何优化
2	增加预计算进行优化
3	增加预计算并且使用 CSA，关键路径也使用 CSA
4	不使用预计算，但是在关键路径使用两次 CSA
5	不使用预计算，在关键路径使用 CSA5

表 2 中 CSA5 是对 CSA 的扩展，其运算如下：

$$S(a,b,c) = a \wedge b \wedge c$$

$$Ca(a,b,c) = [(a \& b) \mid (b \& c) \mid (a \& c)] << 1$$

$$S'(S,Ca,d) = S \wedge Ca \wedge d$$

$$Ca'(S,Ca,d) = [(S \& Ca) \mid (Ca \& d) \mid (S \& d)] << 1$$

$$CSA5(a,b,c,d,e) = CSA(S',Ca',e) = a+b+c+d+e$$

5 种方案都是在全流水线和全展开的基础上再优化实现，实现结果如表 3 所示。

表 3 不同方案资源综合表

Table 3　Different project resources integrated table

对比方案	LUT 消耗（占比）	FF 消耗（占比）	最大频率
1	11768（3.55%）	19488（2.94%）	340MHz
2	11967（3.61%）	26483（4.00%）	420MHz
3	11355（4.42%）	21290（3.21%）	425MHz
4	14809（4.46%）	20100（3.03%）	415 MHz
5	15639（4.72%）	21173（3.19%）	367MHz
最终方案	9415（2.84%）	21514（3.24%）	470MHz

从表中可以看出使用预计算和保留进位加法器结合进行优化与其他类型的优化相比在最高频率方面是最优的，和不进行任何优化相比算法频率提升 130MHz。所有采用预计算的方案与不使用预计算优化相比，其寄存器都有较高消耗。

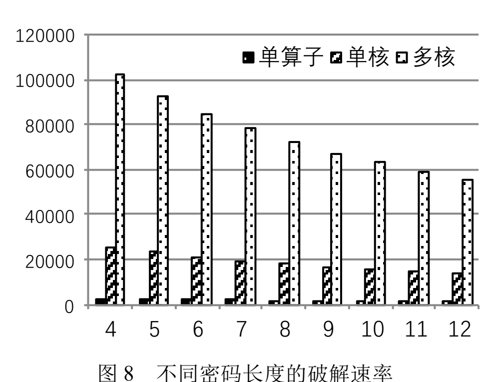


图 8　不同密码长度的破解速率

Fig.8　Different password cracking rate

通过对 SHA-1 算法、数据拼接、以及整体架构优化，并且尽可能的使用芯片资源，最终算法实例化 10 个核心算子模块，占用资源为 75%，实验结果的整体算法优化性能如图 8 所示。

从图 8 中可以看出，在不同密码对应的破解速度在多核 FPGA 板卡上性能是单核的 4 倍，在密码长度为 4 的破译速率达到每秒 102 796 个。通过控制使核与核之间并行运行，且支持多任务。当有大量的加密文档需要恢复时，在每个核运行不同的任务，通过高概率口令的优先穷举，能够很快恢复出其中大部分的口令。与其他文献方法对比，如表 4 所示。

表 4　不同文献破解算法性能表现

Table 4　Different research algorithms performance

对比文献	实验平台（型号）	速率	加速比
AccentRAR	CPU（I7）	1000	1.00
文献 8	GPU（W8000）	10100	10.10
文献 9	GPU（W8000）	17778	17.78
本文	FPGA（XCKU060）	102796	102.80

以软件 AccentRAR 的性能作为基准，本文的性能达到了其 100 倍，比其他文献的方法性能高很多。Hashcat 是一个提供基于 OpenCL 的开源破解软件的网站，其破解性能好，破解算法多等受到大家的广泛使用，因此本文使用 Hashcat 在 GPU（GTX1080）上作验证对比，比较结果如图 9 所示。

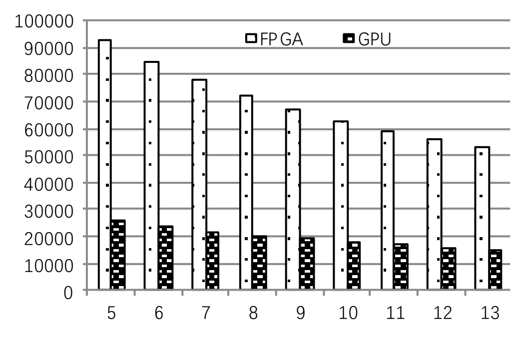


图 9　不同密码长度破译性能对比

Fig.9　Different password length performance

从图中可以看出本文的性能达到了 Hashcat 的 3.5 倍。将本文的优化结果与其他文献、软件对比，实验结果表明了本文的方案有较大的性能提升。

4　结束语

当前 RAR 口令密码破译主要是在 CPU 和 GPU 平台，前者主要的问题是破译速度慢，远远不能满足口令空间的指数级增长，后者由于涉及到访存问题，限制了破译性能，并且功耗较高。针对这些问题，本文提出了多核 FPGA 硬件平台破译的方法，在全流水线架构下采用预计算和保留进位加法器策略缩短关键路径来优化 SHA-1 算法，使 SHA-1 算法的吞吐量达到了 240.64 Gbps。结合 FPGA 的低功耗、并行性、时钟频率高等优势对破解算法的数据拼接和整体架构进行优化，使算法最终的频率达到 200MHz，破解四位密码长度的性能达到每秒 102 796 个，并通过与其他文献进行对比，表明在多核 FPGA 的硬件平台，优化算法后实现的性能有明显的提升，也表明了优化算法方法的可行性。

虽然本文提出的方法使破译性能有了很大的提升，但是对于庞大的口令空间，其计算性能仍然很难满足实际的需求，因此，生成高概率密码结合本文的性能才能更大的提高破译的成功率。而高概率密码则需要结合诸如大数据，统计，人工智能等多方面的知识，而这些将会是破解成功的关键及以后的工作重点。

chinaXiv:201811.00131v1



chinaXiv:201811.00131v1

参考文献：

[1] Wang Ding, Jian Gaopeng, Huang Xinyi, *et al.* Zipf's law in passwords [J]. IEEE Trans on Information Forensics and Security, 2017, 12 (11): 2776-2791.

[2] Ma J, Yang Weining, Luo Min, *et al.* A study of probabilistic password models [C]// Security and Privacy. 2014: 689-704.

[3] Ding Wang, Zhang Zijian, Wang Ping, *et al.* Targeted online password guessing: an underestimated threat [C]// Proc of ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2016: 1242-1254.

[4] 王平，汪定，黄欣沂. 口令安全研究进展 [J]. 计算机研究与发展, 2016, 53 (10): 2173-2188. (Wang Ping, Wang Ding, Huang Xinyi. Advances in password security [J]. Journal of Computer Research and Development, 2016, 53 (10): 2173-2188.)

[5] Bitar A, Cassidy J, Jerger N E, *et al.* Efficient and programmable ethernet switching with a NoC-enhanced FPGA [C]// Proc of the 10th ACM//IEEE Symposium on Architectures for Networking and Communications Systems. New York: ACM Press, 2014: 89-100.

[6] Bayer F M, Cintra R J, Edirisuriya A, *et al.* A digital hardware fast algorithm and fpga-based prototype for a novel 16-point approximate DCT for image compression applications [J]. Measurement Science & Technology, 2012, 23 (11) .

[7] Waidyasooriya H M, Takei Y, Tatsumi S, *et al.* OpenCL-based FPGA-platform for stencil computation and its optimization methodology [J]. IEEE Trans on Parallel and Distributed Systems, 2017, 28 (5): 1390-1402.

[8] 周犇，张云泉，安小景，等. 基于 OpenCL 的 RAR 密码暴力破解算法优化 [C]// 全国高性能计算学术年会论文集. 2014. (Zhou Ben, Zhang Yunquan, An Xiaojing, *et al.* Optimization of RAR password brute-force cracking based on OpenCL [C]// Proc of High-Performance Computing China. 2014.)

[9] An Xiaojing, Zhao Haojun, Ding Lulu, *et al.* Optimized password recovery for encrypted RAR on GPUs [C]// Proc of IEEE International Conference on High Performance Computing and Communications, International Symposium on Cyberspace Safety and Security, and International Conference on Embedded Software and Systems. Washington DC: IEEE Computer Society, 2015: 591-598.

[10] Hu Guang, Ma Jianhua, Huang Benxiong. Password recovery for RAR files using CUDA [C]// Proc of the 8th IEEE International Conference on Dependable, Autonomic and Secure Computing. 2009: 486-490.

[11] Ge Liang, Wang Lianhai. Research of password recovery method for rar based on parallel random search [M]// Applications and Techniques in Information Security. Berlin: Springer, 2014: 211-218.

[12] Dai Zibin, Zhou Ning. FPGA implementation of SHA-1 algorithm [C]// Proc of International Conference on Asic. 2003: 1321-1324.

[13] 李磊，韩文报. FPGA 上 SHA-1 算法的流水线结构实现 [J]. 计算机科学, 2011, 38 (7): 58-60. (Li Lei, Han Wenbao. Implenmentation of pipeline structure on FPGA for SHA-1 [J]. Computer Science, 2011, 38 (7): 58-60.)

[14] Li Bin, Zhou Qinglei, Si Xueming. Mimic computing for password recovery [J]. Future Generation Computer Systems, 2018. 84: 58–77.

[15] 黄谆，白国强，陈弘毅. 快速实现 SHA-1 算法的硬件结构 [J]. 清华大学学报：自然科学版, 2005, 45 (1): 123-125. (Huang Zhun, Bai GuoQiang, Chen Hong-Yi. Efficient hardware architecture for secure hash algorithm SHA-1 [J]. Journal of Tsinghua University: Science and Technology, 2005, 45 (1): 123-125.)

[16] Jiang Liehui, Wang Yuliang, Zhao Qiuxia, *et al.* Ultra high throughput architectures for SHA-1 hash algorithm on FPGA [C]// Proc of International Conference on Computational Intelligence and Software Engineering. 2009: 1-4.

[17] Kim J W, Lee H U, Won Y. Design for high throughput SHA-1 hash function on FPGA [C]// Proc of International Conference on Ubiquitous & Future Networks. 2012: 403-404.

[18] Kakarountas A P, Theodoridis G, Laopoulos T, *et al.* High-Speed FPGA implementation of the SHA-1 hash function [C]// Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. 2007: 211-215.

[19] 雷元武，窦勇，郭松. 基于 FPGA 的高精度科学计算加速器研究 [J]. 计算机学报, 2012, 35 (1): 112-122. (Lei Yuanwu, Dou Yong, Guo Song. High precision scientific computation accumulator on FPGA [J]. Chinese Journal of Campters, 2012, 35 (1): 112-122.)

[20] Monmasson E, Cirstea M N. FPGA design methodology for industrial control systems: a review [J]. IEEE Trans on Industrial Electronics, 2007, 54 (4): 1824-1842.

[21] 谭健，周清雷，斯雪明，等. 全流水架构 MD5 算法在拟态计算机上的实现及改进 [J]. 小型微型计算机系统, 2017, 38 (6): 1216-1220. (Tan Jian, Zhou Qinglei, Si Xueming, *et al.* Implementation and improvement of full-pipeline MD5 algorithm based on mimic compiter [J]. Journal of Chinese Computer Systems, 2017, 38 (6): 1216-1220.)